Improving the Data Quality of Drug Databases using Conditional Dependencies and Ontologies

Olivier Curé

Improving the Data Quality of Drug Databases using Conditional Dependencies and Ontologies

Olivier Curé
Université Paris-Est, LIGM - UMR CNRS 8049, France

Many healthcare systems and services exploit drug related information stored in databases. The poor data quality of these databases, e.g. inaccuracy of drug contraindications, can lead to catastrophic consequences on the health condition of patients. Hence it is important to ensure their quality in terms of data completeness and soundness.
In the database domain, standard Functional Dependencies (FDs) and INclusion Dependencies (INDs), have been proposed to prevent the insertion of incorrect data. But they are generally not expressive enough to represent a domain specific set of constraints. To this end, conditional dependencies, i.e. standard dependencies extended with tableau patterns containing constant values, have been introduced and several methods have been proposed for their discovery and representation. The quality of drug databases can be considerably improved by their usage.
Moreover, pharmacology information is inherently hierarchical and many standards propose graph structures to represent them, e.g. the Anatomical Therapeutic Chemical classification (ATC) or OpenGalen's terminology. In this paper, we emphasize that the technologies of the Semantic Web are adapted to represent these hierarchical structures, i.e. in RDFS and OWL. We also present a solution for representing conditional dependencies using a query language defined for these graph oriented structures, namely SPARQL. The benefits of this approach are interoperability with applications and ontologies of the Semantic Web as well as a reasoning-based query execution solution to clean underlying databases.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications-Data mining

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Data quality, Description logics, conditional dependencies

1. INTRODUCTION

The quality of information stored in databases has a significant impact on the efficiency of organizations and businesses managing and exploiting them [Batini and Scannapieco 2006]. This is particularly relevant in the medical domain where inaccurate or false information can have dramatic effects on the health of patients. In this paper, we consider a self-medication application [Curé 2004] that we have designed with domain experts and which is being proposed to clients of several major French insurance companies.

At the core of this application is a set of a relational databases containing information on symptoms of the self-medication domain as well as drugs available on the French market. The drug database contains all the information present in the Summary of Product Characteristics (SPC) and proposes additional information such as a rating based on an efficiency/tolerance ratio [Giroud and Hagege 2001], price, opinion from domain experts, social security system reimbursement rate, etc. Our databases also integrate as much information as possible on international standards like the Anatomical Therapeutic Chemical classification (ATC) and the tenth

edition of the International Statistical Classification of Diseases and Related Health Problems (ICD 10), etc. This integration supports the representation of most of the explicit information required to perform the needed inferences.

This application's main functionality consists in proposing a list of over-the-counter (OTC) drugs based on a patient's anamnesis. Note that this form of diagnostic does not always end up with a drug proposition. In fact, if the symptom described and/or the patient profile are not adapted to self-medication then the end-user is advised to consult a healthcare professional. A patient profile stores general information on the gender, age and health condition (e.g. pregnancy, breast feeding) as well as known diseases, allergies and current drug treatments, all encoded using standard terminologies. This application targets the general public, i.e. end-users assumed to have no medical knowledge. Hence we can not expect end-users to detect false information, as it could be the case for an application designed for health professionals. Consequently, the soundness and completeness of the stored information is of a high priority as inaccurate information may be proposed to the patient and jeopardize their levels of health.

The characteristics of our drug database imposes to use an automatic or semi-automatic data quality tool. They correspond to the storage of a SPC and additional information for ten thousand drug products and a high update rate, i.e. modifications of existing drugs, insertions of new products and removal of old ones. These operations are generally performed on a weekly basis. Although the data quality market already proposes some effective tools, they generally involve intensive interactions with domain experts and are based on a limited set of constraints found in relational databases. Recently, new forms of data dependencies have been investigated and their relevance in data exchange/integration as well as data quality have been emphasized. Some of them are based on extensions of the notion of data dependency which has been thoroughly investigated and exploited in the domain of relational databases [Abiteboul et al. 1995], e.g. Functional and INclusion Dependencies (henceforth denoted FDs and INDs). For instance, [Fan 2008] highlights an attempt to improve data quality using conditional dependencies. In principle, these dependencies hold only for the tuples that satisfy some conditions. So far, two forms have been investigated: Conditional Functional Dependencies (CFDs) [Bohannon et al. 2007] and Conditional INclusion Dependencies (CINDs) [Bravo et al. 2007]. They correspond to conditional counterparts of respectively FDs and INDs. Moreover, they are supposed to capture more of the inconsistencies of real-life data and can help to implement efficient data cleansing tools.

This paper concentrates on the data quality issues of our drug database. Central to our solution is the ATC classification which divides drug molecules into different groups according to the organ or system on which they act and/or their therapeutic and chemical characteristics. In the next example, we consider a simplified version our database and the peculiar aspect of cleansing the molecule contraindication information by integrating CFDs and CINDs. Note that in our application, this approach is also used for other drug related information such as side-effects, disease contraindications, etc.

Example 1.1: Consider the following drug database schema:

drug (CIP, DRUGNAME, DRUGRATE, DRUGTYPE)

atc (ATCCODE, ATCNAME)
atcDrug (CIP, ATCCODE)
drugContra (CIP, ATCCODE)

where the *drug* relation contains a (French) product identifier (CIP), a product name (DRUGNAME), the reimbursement rate (DRUGRATE) and the type of the product (DRUGTYPE) corresponding to allopathy, homeopathy, etc. The *atc* relation stores all ATC codes (ATCCODE) and their names (ATCNAME). Another relation, *atcDrug*, relates drug products to their ATC codes. Finally, the relation *drugContra* stores the ATC codes a drug is contraindicated to. Some of the FDs and INDs that hold for this database include:

fd1 : *drug* (CIP → DRUGNAME)
ind1: *drugContra*(ATCCODE) ⊆ *atc*(ATCCODE)
ind2: *atcDrug*(CIP) ⊆ *drug*(CIP)

Table I presents a portion of our drug database instance. Starting from this small extract, we observe that standard FDs and INDs are not sufficient to represent the set of constraints associated to the pharmacology domain. For example, we would like to state that drug reimbursement at a 65% rate must be of the type 'allopathy' and that only allopathic drugs are present in the *atcDrug* relation. Moreover, we would like to represent the fact that all drug products contraindicated to *Iproniazid* must be either composed of the molecules *Zolmitriptan* or *Dextromethorphan*. These constraints require a notion of condition, a feature not supported by traditional FDs and INDs. But their conditional counterparts, resp. CFDs and CINDs, have been introduced to support these constraints:

cfd2: *drug* (DRUGRATE=65 → DRUGTYPE='allopathy')
cind3: *atcDrug*(CIP) ⊆ *drug*(CIP, TY='allopathy')
cind4: *drugContra*(CIP, ATCCODE='N06AF05') ⊆
atcDrug(CIP,ATCCODE='N02CC03' || 'R05DA09')

The database instance of Table I does not satisfy these 3 conditional dependencies. For instance, cfd2 is violated by tuple t6 since the *Cephyl* drug is a homeopathic drug but is reimbursed at the 65% rate. Moreover cind3 is violated by tuple t12 because as a homeopathic drug, *Cephyl* should not have an entry in the *atcDrug* relation. The treatment of cind4 is more involved since it implies a form of disjunction. That is a drug contraindicated with *Iproniazid* must be either composed of the *Dextromethorphan* or the *Zolmitriptan* molecules. This is not the case for tuple t16 since the drug identified by (CIP) value 3786018 is composed of *Phloroglucinol*. Interestingly, a similar form of disjunction has recently been introduced in the context of CFDs, yielding the notion of eCFDs [Bravo et al. 2008]. Besides, we would also like to detect that the *Tuxium 30mg* drug (t4), which contains the *Dextromethorphan* molecule (t10), is not contraindicated to the *Iproniazid* molecule.

Most of the investigations conducted on conditional dependencies have focused on their syntax, semantics and properties on precise problems, e.g. consistency, implication and existence of finite axiomatization. The understanding of these issues now offers an ideal environment for the design of efficient data cleansing techniques and tools. In order to address these issues, a first step consists of discovering conditional dependencies. This effort is necessarily automated since manual discoveries necessitate an intensive involvement of domain experts and are error-prone due to

Table I. Records from the drug database

Tuple	CIP	DRUGNAME	DRUGRATE	DRUGTYPE
$t1$	3533665	$Zomigoro2.5mg$	65	*Allopathy*
$t2$	3282358	$Capsyl15mg$	0	*Allopathy*
$t3$	3544309	$Zomig2.5mg$	65	*Allopathy*
$t4$	3311692	$Tuxium30mg$	35	*Allopathy*
$t5$	3786018	$PhloroglucinolArrow$	35	*Allopathy*
$t6$	3187559	$Cephyl$	65	*Homeotherapy*

(a) *drug* relation

Tuple	CIP	ATCCODE
$t7$	3533665	$N02CC03$
$t8$	3282358	$R05DA09$
$t9$	3544309	$N02CC03$
$t10$	3311692	$R05DA09$
$t11$	3786018	$A03AX12$
$t12$	3187559	$N02BA01$

(b) *atcDrug* relation

Tuple	CIP	ATCCODE
$t13$	3533665	$N06AF05$
$t14$	3282358	$N06AF05$
$t15$	3544309	$N06AF05$
$t16$	3786018	$N06AF05$

(c) *drugContra* relation

Tuple	ATCCODE	ATCNAME
$t17$	$R05DA09$	*Dextromethorphan*
$t18$	$N06AF05$	*Iproniazid*
$t19$	$N02CC03$	*Zolmitriptan*
$t20$	$A03AX12$	*Phloroglucinol*
$t21$	$N02BA01$	*Acetylsalicilcacid*

(d) *atc* relation

large volumes of information. Automated methods based on the analysis of sample database instances have already been published for CFDs in [Chiang and Miller 2008], [Golab et al. 2008] and [Fan et al. 2009] but are missing for CINDs. In this paper, we present an algorithm for the discovery of CINDs. Hence both CFDs and CINDs can be discovered from a given dataset, leading to the next issue: finding representations for efficient detection of their violations. So far, all proposed solutions opted for an SQL query formalism. In this work, we consider another representation which is more adapted to the graph-based hierarchical structures that one can find in standards such as the ATC classification, ICD 10 or OpenGalen. We generalize this notion of a hierarchical structure as an ontology and argue that this is ideal for the kind of inferences that we would like to perform on our self-medication application. Moreover, we will show that the representation of discovered CFDs and CINDs will be more compact using a query language adapted to an ontology structure.

Contributions

The first contribution of this work is to propose a sound and complete algorithm for the discovery of CINDs from a relational database instance.

A second contribution consists in detecting conditional dependencies violations in a relational database from a set of queries expressed at the ontology level, using the DBOM (DataBase Ontology Mapping) tool [Curé and Bensaid 2008]. This

approach enables to detect the potential violations of conditional dependencies by executing SPARQL queries, a W3C recommendation for querying RDF (Resource Description Framework) data.

Finally, we present an evaluation on different datasets. In the first hand, our CIND discovery solution is evaluated on synthetic databases. Then our data quality solution is tested on our drug database and we emphasize that an important amount of information anomalies are automatically detected and semi-automatically repaired , i.e. requiring interactions with domain experts.

The paper is organized as follows. Section 2 provides some related work in the domain of conditional dependencies and discovery of inclusion dependencies. Section 3 presents the basic notions related to conditional dependencies, the ATC classification and ontologies. In Section 4, we provide an algorithm to discover CINDs. In Section 5, we propose SPARQL-based solutions to detect CFDs and CINDs violations. An experimental study is provided in Section 6. Section 7 considers some implications for researchers and practitioners who aim to apply the data quality solution proposed in this paper. We conclude and present perspectives in Section 8.

2. RELATED WORK

To the best of our knowledge, this paper is a first approach to tackle data quality using ontologies and database constraints in the healthcare domain. Hence, in this section we present related work on the issues of discovering data dependencies and detect potential violations. As stated previously in this paper, most of the work published on these issues concern CFDs.

Three papers have recently been published on discovering CFDs. In [Golab et al. 2008], the authors investigate the computational complexity, i.e. NP-complete, of the automatic generation of optimal tableaux for a fixed traditional FD, i.e. generating CFDs. Then they provide efficient heuristics to discover approximation from a sample database. [Chiang and Miller 2008] discovers traditional FDs and patterns in CFDs. The discovery algorithm uses a level wise approach which may not scale well with large database instances. Particularly interesting in this paper are the investigation of three pertinent measures for discovered CFDs. These measures correspond to *conviction*, χ^2 and *support* which is also exploited in this work. Finally [Fan et al. 2009] proposes several algorithms to discover CFDs. These algorithms distinguish between constant CFDs, i.e. patterns filled with constant values, and general CFDs, i.e. variables and constant are allowed. Concerning general CFDs, two methods are presented: a level wise and a depth first search approach. In contrast to [Fan et al. 2009], our approach discovers data dependencies when the embedded traditional dependencies are given.

The work of [Goethals et al. 2008] mines association rules of the form $Q_1 \Rightarrow Q_2$ where Q_1 and Q_2 are simple conjunctive queries and Q_2 is contained in Q_1. The CINDs investigated in this paper can be considered as association rules with a confidence of 100%. Thus the technique proposed in [Goethals et al. 2008] is able to discover CINDs. Moreover an investigation is needed on the types of CINDs that can be generated with this algorithm.

[Bohannon et al. 2007] is the only investigation on detecting violations of con-

ditional data dependencies. This approach is limited to CFDs and proposes techniques based on the execution of two SQL queries. A first query finds single-tuple violations, i.e. inconsistent tuples based on dissimilarities between constants in the tuples of a database instance and T_p patterns. The second query finds multi-tuple violations, i.e. several tuples in the database instance match the LHS (Left Hand Side) of attributes of a pattern of T_p but concerning the RHS (Right Hand Side) attributes of the CFDs, the instance tuples do not match. This two step process is due to the possibility for a set of CFDs to be inconsistent. Comparatively, one of our solutions also generates SQL queries to detect violations of CINDs. The main difference with the approach of [Bohannon et al. 2007] lies in the fact that we only need one query to find violations. This is due to the property that a set of CINDs is always consistent. In this same paper, the authors propose an efficient solution to detect violation of multiple CFDs. This technique is based on a merge of the CFDs and a query generation.

3. PRELIMINARIES

3.1 Conditional dependencies

In this section, we present the syntax and semantics of CFDs and CINDs which are respectively extensions of FDs and INDs with patterns.

A CFD ψ defined on a database relation R is a pair $R(X \rightarrow Y, T_p)$ where X and Y are attribute sets in R, $X \rightarrow Y$ is a standard FD [Abiteboul et al. 1995] and T_p is a pattern tableau containing all attributes in X and Y. For each attribute A in $(X \cup Y)$, the value of the attribute A of the tuple t_p in T_p, denoted $t_p[A]$ is either a value from the domain of A or a variable denoted by '_'. A tuple t matches a tuple t_p in T_p if for each attribute A in T_p, for some constant 'a', $t[A] = t_p[A]$ ='a' or $t_p[A]$ ='_'.

An instance D of R satisfies the CFD ψ, denoted $D \models \psi$, if for every pair of tuples t_1 and t_2 in D, for each pattern tuple t_p in T_p and for every attribute A in X, if $t_1[A] = t_2[A]$ and both t_1 and t_2 match $t_p[A]$, then $t_1[B] = t_2[B]$ and t_1, t_2 both match $t_p[B]$ for each attribute B in Y.

A CIND ϕ defined over a pair of relations R_1 and R_2 is a pair $(R_1(X; X_p) \subseteq R_2(Y; Y_p), T_p)$ where X, X_p and Y, Y_p are attribute sets of R_1 and R_2 respectively, $R_1(X) \subseteq R_2(Y)$ is a standard IND [Abiteboul et al. 1995] and T_p is a pattern tableau of ϕ with attribute sets X_p and Y_p such that each pattern tuple t_p and each attribute B in X_p (or Y_p), $t_p[B]$ is a constant in the domain of B.

An instance (D_1, D_2) of (R_1, R_2) satisfies the CIND ϕ, denoted $(D_1, D_2) \models \phi$, iff for each tuple t_p in T_p and for each t_1 in D_1, if $t_1[X_p] = t_p[X_p]$, then there must exist t_2 in D_2 such that $t_1[X] = t_2[Y]$ and $t_2[Y_p] = t_p[Y_p]$.

Example 3.1: The constraints cfd2, cind3 and cind4 of our running example are respectively represented by the following conditional dependencies:
ψ_2 = drug(DRUGRATE $\rightarrow$ DRUGTYPE, T_2) with T_2 = {(65,'allopathy')}
ϕ_3 =(atcDrug[CIP] $\subseteq$ drug[CIP;DRUGTYPE], T_3) with T_3={('_','_','allopathy')}
ϕ_4 =(drugContra[CIP;ATCCODE] $\subseteq$ atcDrug[CIP;ATCCODE], T_4) with T_4 = {('_','N06AF05','_',{'N02CC03','R05DA09'})}

We say that a database instance I satisfies a set Ψ of CFDs and Φ of CINDs, if

$I \models \phi$ for all ϕ of Φ and $I \models \psi$ for all ψ of Ψ.

3.2 ATC classification

The ATC system (http://www.whocc.no/atcddd/) is an international classification of drugs and is part of WHO's initiatives to achieve universal access to needed drugs and rational use of drugs. In this classification, drugs are organized in groups at five different levels. The first level of the code is based on a letter for the anatomical group and defines 14 groups. The second level corresponds to a pharmacological/therapeutic subgroup. The third and fourth levels are chemical/pharmacological/therapeutic subgroups. Finally, the fifth level corresponds to chemical substances and supports the classification of drugs according to Recommended International Non-proprietary Names (rINN).

We now provide an extract from the ATC hierarchy for some cough suppressants:

```
R: Respiratory system
 R05: Cough and cold preparations
   R05D: Cough suppressants, excluding combinations with expectorants
     R05DA: Opium alkaloids and derivatives
       ...
       R05DA08 Pholcodin
       R05DA09 Dextromethorphan
       ...
       R05DA20 Combinations
```

The *R05DA20* code identifies compound chemical products combining *opium alkaloids* with other substances. The *Hexapneumine* syrup is generally classified with this code but we find that this approach is inaccurate since it does not quantify and qualify the contained molecules. We argue for another approach where this drug is classified by the conjunction of its compounds, i.e. *pholcodin*, *chlorphenamin* and *biclotymol* which respectively correspond to R05DA08, R06AB04 and R02AA20. Thus we obtain a more accurate classification which is exploited in the different inferences performed in our system and the representation of the conditional dependencies toward improving the overall data quality of the databases.

3.3 Ontology

In information technology, an ontology provides a shareable and reusable piece of knowledge about a specific domain. In such a context, an ontology is a set of concepts and their relationships, that are specified more or less formally in order to create an agreed-upon vocabulary. The ATC hierarchy typically corresponds to this definition of an ontology. In our solution, we have selected Description Logics (DL) [Baader et al. 2003] as a mean to represent ontologies. This family of knowledge representation formalisms allows to represent and reason over domain knowledge in a formally and well-understood way. Central DL notions are concepts (unary predicates), relationships, also called roles or properties (binary predicates) and individuals. A standard DL knowledge base is usually defined as $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ (or TBox) and $\mathcal{A}$ (or ABox) consist respectively of a set of concept descriptions (resp. concept and role assertions).

One of the goals of an ontology is to support the integration and sharing of

data across different applications and organizations. A first step toward reaching this goal is to establish a closer cooperation with a robust technology to deal with large volumes of data, e.g. relational DBMS. We have designed the DBOM tool especially for this task: it integrates data stored in relational database instances into a knowledge base compliant with Semantic Web standards (RDF Schema or the Web Ontology Language (OWL)). In this context, DBOM proposes a solution to the impedance mismatch problem by proposing a mapping language that allows to specify how to transform data retrieved from tuples of the databases into objects of the target knowledge base. DBOM also handles inconsistencies that may be caused by the execution of uncertain mappings. With such a system, the generation of an OWL ontology corresponding to our drug databases requires the definition, through a graphical user interface, of few mapping assertions.

Example 3.2: Considering the database of Example 1.1, we can define an ontology where the classification of ATC is represented as a hierarchy of concepts, e.g. the *R05DA09* concept is a subconcept of the *R05DA* concept, denoted $R05DA09 \sqsubseteq R05DA$. Similarly for this classification branch, the following concept assertions are created: $R05DA \sqsubseteq R05A$, $R05D \sqsubseteq R05$, $R05 \sqsubseteq R$ and $R \sqsubseteq ATC$. Each sub concept of the ATC concept has two data type properties: *atcCode* and *atcName* which respectively relate an ATC concept to its code and a label. A single *Drug* concept is created and the following data type properties are attached to it: *drugCip, drugName, drugRate* and *drugType*. Each drug in the database instance becomes an individual in the knowledge base with a type corresponding to *Drug*. Finally, the *drugContra* and *atcDrug* relations are transformed into object properties, respectively *contraIndicated* and *contains*, which relate a *Drug* to ATC individuals.

Finally, a query solution is needed to retrieve information from our ontologies. For this purpose, the W3C has published a graph-matching query language called SPARQL (Sparql Protocol And RDF Query Language). RDF (Resource Description Framework) is a directed, labeled graph data format which is composed of triples, i.e. subject, predicate and object. The ontology generated with DBOM follows this format. In this context, a SPARQL query consists of a pattern which is matched against an RDF graph and the values obtained from this matching are processed to give the answer. Such a query has three parts: (i) pattern matching which includes several interesting features, e.g. filtering, (ii) solution modifiers, e.g. distinct, order, limit and (iii) the output of the query.

We have now presented all the machinery needed for our data quality approach. The general principle is the following: data from our databases are integrated in the knowledge, via DBOM mapping assertions, and data dependencies are automatically represented with SPARQL queries. The execution of these queries will highlight inconsistent individuals of the knowledge base. These individuals can be easily transposed to database tuples, using the DBOM mapping assertions. Note that in the context of our self-medication application, the knowledge base also serves at runtime to perform various inferences.

We now need to discover CFDs and CINDs that hold over a database instance. Since such solutions have already been presented and validated for CFDs, the next section focuses on the first solution to discover CINDs.

4. CINDS DISCOVERY

Given an instance database D of R_1, a CIND discovery algorithm searches for CINDs that hold on D. Obviously, for a CIND ϕ, we are expecting the size of the pattern tableau T_p to be inferior to the number of tuples of D. This relationship implies that redundant, trivial and pseudo-trivial [Mitchell 1983] CINDs are removed from the final set of discovered CINDs. Hence, our method aims to discover a canonical cover of a set of CINDs. That is a minimal set of CINDs, containing non trivial and non redundant dependencies, from which all CINDs on D can be derived via implication analysis. We recall that a sound and complete inference system for CINDs has been detailed in [Bravo et al. 2007].

Moreover, when mining the database instance, we only consider patterns that are supported by a minimum amount of tuples, i.e. the number of tuples motivating a pattern needs to be greater or equal to a threshold denoted by the *support*.

Definition: The *support* of a query Q executed over an instance D is the number of distinct tuples in the answer of Q on D. A query is said to be *frequent* in D if its support exceeds a given minimal support threshold.

The task of setting the support threshold is the responsibility of the end-user. Obviously, the higher the support, the less CIND patterns are discovered.

The algorithm we present to discover CINDs start from a set of approximated INDs, that is IND "almost" holding on some relations R_1 and R_2, and which are computed using the method proposed in [Marchi and Petit 2003]. The inputs of this algorithms is a database instance and a set of INDs. Several data structures are exploited in this algorithm. To summarize, a list of INDs (*indList*) stores all the information related to an IND of the following form: $R_1[X] \subseteq R_2[Y]$, similarly *cindList* stores a list of CINDs of the form: $(R_1[X; X_p] \subseteq R_2[Y; Y_p], T_p)$. Associated to these structures are methods developed to retrieve and insert data, e.g. retrieving the list of attributes in ϕ or inserting a new entry in a pattern tableau.

Example 4.1: In this example, we consider: (1) the binary IND ind5: $R[r1, r2] \subseteq S[s1, s2]$, (2) a support threshold of *2* and (3) the following database extract:

r1	**r2**	**r3**	**r4**
1	1	a	x1
1	2	a	x2
1	3	a	x3
2	1	x4	d
2	2	x5	d
2	3	x6	d
2	4	x7	d

(a) relation R

s1	**s2**	**s3**	**s4**	**s5**
1	1	b	c	y1
1	2	b	c	y2
1	3	b	c	y3
2	1	y4	e	y6
2	2	y7	e	y8
2	3	y9	f	y10
2	4	y11	e	y12

(b) relation S

Our algorithm, *discoverCIND* is presented in Algorithm 1. In principle, it creates a empty CIND structure (line 1) and for each IND in a list of INDs, it searches for attribute candidates from both the LHS and RHS of the current IND (lines 2-8). Line 9 makes sure that we are not searching patterns between pairwise attributes of X and Y, e.g. between X^i and Y^i for $1 \leq i \leq |X|$. Then an SQL query is generated (line 11) and the *update* function is invoked (line 12). Finally, line 18 passes the CIND structure to the *minimalCover* function.

Algorithm 1 discoverCIND

Input: *db* a Database Connexion, *indList* a list of INDs
Output: *cindList* a list of CINDs

```
1: cindList ← ∅
2: for all ind ∈ indList do
3:   lhsCandList ← retrieve(ind, R1, X)
4:   if lhsCandList.size > 0 then
5:     for all lhsCand ∈ lhsCandList do
6:       rhsCandList ← retrieve(ind, R2, Y)
7:       if rhsCandList.size > 0 then
8:         for all rhsCand ∈ rhsCandList do
9:           if ¬(containedIn(lhsCand, rhsCand, ind) then
10:            qry ← genQuery(lhsCand, rhsCand, ind)
11:            update(cinds, qry)
12:          end if
13:        end for
14:      end if
15:    end for
16:  end if
17: end for
18: return minimalCover(cindList)
```

The goal of the *retrieve* function is to return a set of candidate attributes from a given relation. These candidate attributes are later mined to search patterns. This function is being called for both the LHS and RHS relations of a IND. Intuitively, the function stores all attributes of a relation in a list and returns it.

Example 4.2: In the context of Example 4.1, *retrieve* returns for *ind5*: *lhsCandidate* = {r1,r2,r3,r4}, *rhsCandidate*={s1,s2,s3,s4,s5}

The *genQuery* function generates an SQL query of the following form:

select distinct r_1.lhsCan, r_2.rhsCan **from** R_1 r_1 R_2 r_2
where $r_1.X^1 = r_2.X^1$ **and** .. **and** $r_1.X^n = r_2.X^n$
group by r_1.lhsCan, r_2.rhsCan **having count(*)** $\geq$ support;

Essentially, this query searches for groups of value pairs on elements in *lhsCandidate* and *rhsCandidate* that match on the values of X and Y. Additionally, the number of tuples from each group needs to be superior or equal to the support parameter.

Example 4.3: For our running example, the first query generated with *genQuery* is presented below and its answer set corresponds to the tuple (a, b):

select distinct R.r3,S.s3 **from** R, S
where S.s1 = R.r1 AND S.s2 = R.r2
group by S.s3,R.r3 **having count(*)** $\geq$ 2;

The *update* function takes as input the query generated by the *genQuery* method, executes it and if the answer set is not empty, stores the resulting pairs in a CIND structure. Hence, at this step of execution, this structure stores all the pairs of patterns discovered for a given IND.

Example 4.4: Mining the sample database yields the following CINDs pattern tableau (in tabular form for readability reasons):

r1	r2	r3	r4	s1	s2	s3	s4	s5
_	_	a	_	_	_	{b}	_	_
_	_	a	_	_	_	_	{c}	_
_	_	_	d	_	_	_	{e}	_
_	_	_	d	_	_	_	{f}	_

The last step required for the generation of CIND is to regroup these pattern pairs such that the final set of CINDs is minimal. These operations are performed by the *minimalCover* function presented in Algorithm 2. The method *exist* checks if the *lhs* value (respectively *rhs*) are present in a pattern of $cindList'$. The method *add* adds a new entry in $cindList'$ with the *lhs* value (resp. of *rhs*) at the proper place (attributes stored in *lhs* and *rhs*).

The *organize* function is invoked if for a given attribute and relation couple, the value in *lhs* or *rhs* is already stored somewhere in $cindList'$, we denote this a *match*. This algorithm marks tuples until a co-occurence situation is discovered or all tuples have been analyzed. The *merge* function searches for a way to store the information contained in the *lhs* and *rhs* variables into $cindList'$. Once a *match* is discovered, the algorithm analyzes the counterpart of IND, e.g. *rhs* is matched on *lhs*. The operations performed are different whether the match is found on the *lhs* or *rhs*. For the LHS, the algorithm searches if the current pattern of $cindList'$ contains a value at the *rhs.Attribute* position. If this is not the case then the value of *rhs* is added to the list of values for this attribute in $cindList'$, i.e. case of disjunction, otherwise the *co-occurence* method is called. This function generates a query to discover if values of $cindList'$ on the RHS co-occur with the *rhs* value at *rhs.attribute*. The generated SQL query in the case of LHS match has the following form (a similar query is generated for RHS matches):

Algorithm 2 minimalCover

Input: *cindList* list of CINDs
Output: $cindList'$ list of CINDs

1: $cindList' \leftarrow \emptyset$
2: **for all** $cind \in cindList$ **do**
3: $lhs \leftarrow$ retrieve attribute and value on LHS
4: $rhs \leftarrow$ retrieve attribute and value on RHS
5: **if** $\neg$exist(lhs, rhs) **then**
6: add$(lhs, rhs, cindList')$
7: **else**
8: organize$(lhs, rhs, cindList')$
9: **end if**
10: **end for**
11: **return** *cindList*

select 1 from R_1 r_1 R_2 r_2

where $r_1.X^1 = r_2.Y^1$ **and** .. **and** $r_2.Y^1 = t_p.Y^1$ **and**
for all entries in the current LHS pattern of $cindList'$ with a constant value, generate a conjunction of $attribute = value$

group by for all entries in the current pattern of RHS $cindList'$ with a constant value, generate a conjunction of *attributes*

having count(*)$\geq$ support **and** for all entries in the current RHS pattern of $cindList'$ with a constant value, generate a conjunction of $attribute = value$ pairs;

Example 4.5: We illustrate an SQL query generated by the *organize* function in the context of our running example:

select 1 from R,S **where** r1=s1 **and** r2=s2 **and** r3='a'
group by s3,s4 **having count(*)**$\geq$ 3 **and** s3='b' and s4='e' ;

If the execution of this query returns a non empty answer, then the patterns are merged by the *merge function*. For matches on the RHS, the operations performed are similar but a main difference is the absence of disjunctions. This is due to the syntax restriction of the embedded IND of a CIND.

Example 4.6: This example presents 2 remarkable situations: creation of a disjunct and merging of patterns. The minimal CINDs generated by function *organize* corresponds to: ϕ_5: ($R[r_1, r_2; r_3, r_4] \subseteq S[s_1, s_2; s_3; s_4]$,$T_3$), with T_3 (in tabular form for readability reasons):

r1	r2	r3	r4	s1	s2	s3	s4
_	_	a	_	_	_	{b}	{c}
_	_	_	d	_	_	_	{e,f}

In the next section, we emphasize that the expressiveness extension of CINDs (due to the introduction of disjunction in the RHS) and the possibility to merge tableaux is quite useful in detecting violations.

5. DETECTING CFD AND CIND VIOLATIONS

The CFDs and CINDs discovered using respectively the algorithms of [Fan et al. 2009] and our method can now be integrated into our data quality solution. A first step toward cleansing erroneous and inconsistent data is to identify responsible tuples. The main objectives of our solution are accuracy, i.e. precise identification of the dirty tuples, execution of a minimal set of queries to detect inconsistencies, and efficient runtime performances. Remember that we are now dealing with a knowledge base rather than a relational database. Hence, the queries we will generate from the discovered conditional dependencies are serialized in SPARQL. The main motivation toward representing conditional dependencies through queries is their declarative properties, e.g. easier to share and maintain, compared to a procedural approach, i.e. hard coded in a programming language. The execution of these queries on the knowledge base returns a set of individuals that are at the source of dirty data.

The setting is the same for all query generation: a knowledge base K and sets Σ of conditional dependencies where $\Sigma = \Phi \cup \Psi$, with Ψ and Φ sets of respectively CFDs and CINDs. Intuitively, we want to identify all inconsistent ontology concepts which are violating Σ. To do so, we consider the concept and individual generation

approach used in DBOM which consists in concatenating the name of the concept and the value of the primary key of the tuple. For instance, for the *Tuxium* drug whose *cip* is 3311692, a *Drug_3311692* node is created in the knowledge base graph. We are using this representation to identify object causing an inconsistency with respect to our conditional dependencies. Note that DBOM supports compound primary keys, i.e. primary keys composed of several attributes, and our tuple identification system takes benefit of it.

The mapping between database and ontology entities is exploited by our query generation solution. We consider a mapping function M relating elements from the relational database to elements of the ontology. For instance, for a relation R, the corresponding ontology concept is M(R) and similarly for both types of properties. Thus, in the context of our self-medication application, a GUI displays inconsistent concepts with the associated database relation and primary key value(s). Moreover, an explanation is provided for each inconsistency.

For the sake of simplifying the SPARQL queries generated by our methods, we consider that pattern tableaux are singletons, i.e. they contain a single tuple. Note that the methods we will present also apply when several tuples are present in the tableaux. Moreover, we consider that the elements of a pattern tableau can be accessed like an associative array. For instance, consider the following CFD: $R(\{x_1, x_2\} \rightarrow \{y_1, y_2\}, \{('a', 'b', 'c', 'd')\})$, then $T_p[x_2]$ returns the value 'b'.

Concerning CFDs and CINDs, we tackle the following issues: the efficient detection of CFD and CIND violations and the less frequently addressed issue, at least using conditional data dependencies, of missing values. We propose SPARQL query generations for both of these approaches. The SPARQL queries that our system generates returns a single identifier in the SELECT clause and can contain the following in the WHERE clause: (i) triples with variables (staring with a '?' symbol, predicate and concept names, either starting with a 'rdf:' namespace or a ':' symbol to indicate that it tackles the current ontology; (ii) FILTER operations which enable to test the equivalence of a variable with a data value and (iii) OPTIONAL with FILTER and bound patterns to support a negation as failure approach. This approach does not force all the query pattern to hold. Finally, different triple patterns are generated whether a database relation is mapped to a concept, e.g. *drug* to *Drug*, or to an object property, e.g. *actDrug* to *contains*. Typically, a mapping to a concept will generate a triple of the form: $?x\ rdf : type\ M(R)$, while a mapping to an object property will generate: $?x : M(R)\ ?y$.

5.1 CFD-based detection of inconsistent data

Remember that a CFD ψ, defined on a relation R, is a pair $R(X \rightarrow Y, T_p)$ where X and Y are non empty sets of attributes. Using the database/ontology mapping, we can translate this CFD into the following SPARQL query:

(1) SELECT ?x WHERE { ?x rdf:type M(R).
(2) ?x :M(X_i) ?x_i. FILTER (?x_i = $T_p[x_i]$).
(3) ?x :M(Y_i) ?y_i. FILTER (?y_i != $T_p[y_i]$). }

where line 1 displays the identity of the inconsistent knowledge base individual. Lines 2 and 3 correspond to triple patterns which are matched to our knowledge

base graph. The system generates lines 2 for each x_i in X that has a non-empty $T_p[x_i]$. Similarly, for each y_i in Y that has a non empty $T_p[y_i]$, lines 3 are generated.

Example 5.1: For ψ_2 (drug(DRUGRATE $\rightarrow$ DRUGTYPE, T_2) with T_2 = {(65,'allopathy')}), the following SPARQL query is generated:

```
SELECT ?drug WHERE { ?drug rdf:type :Drug.
       ?drug :drugRate ?rate. FILTER (?rate=65).
       ?drug :drugType ?type. FILTER (?type != "allopathy").}
```

Intuitively, this query searches for all individuals that are of type *Drug*, whose rate is equal to 65% and whose drug type is not equal to 'allopathy'. The execution of this query on the knowledge base generated from Table I's dataset will return the object *Drug_3187559* since this drug is reimbursed at a 65% rate but is a homeopathic drug.

5.2 CIND-based detection of inconsistent data

We distinguish two kinds of queries generated for the CIND detection issue: one for the non-disjunctive case and another handling disjunctions in Y_p. Both of these queries involve the use of OPTIONAL together with FILTER and bound patterns. We now propose examples of generated queries on our running example.

Example 5.2: Disjunction-free queries
The SPARQL query for cind3 (*atcDrug*(CIP) $\subseteq$ *drug*(CIP, TY='allopathy')) is:

```
SELECT ?drug WHERE { ?drug :contains ?atc.
    OPTIONAL { ?drug :drugType ?type.   FILTER (?type = "allopathy").}
    FILTER(!bound(?type)).}
```

In essence, this query searches for all drug individuals that are related to at least one ATC code and which have a drug type different to allopathy or no drug type expressed. This query returns the individual identifier corresponding to the *Cephyl* drug since it is a homeopathic drug but is present in the *atcDrug* relation.

Example 5.3: Queries with disjunctions
For cind4 (*drugContra*(CIP, ATCCODE='N06AF05')$\subseteq$*atcDrug*(CIP,ATCCODE= 'N02CC03' || 'R05DA09'), the generated SPARQL query is:

```
SELECT ?drug WHERE { ?drug rdf:type :Drug.
  ?drug :contraIndicated ?atc. ?atc :atcCode "N06AF05".
  OPTIONAL {?drug :contains ?atc2. ?atc2 :atcCode ?code.
     FILTER (?code = "R05DA09" ||  ?code="N02CC03").}
  FILTER(! bound(?atc2)).}
```

The pattern of the query is similar to the one in Example 5.2 except that it introduces a disjunction symbol (i.e. '||') and an additional triple in the OPTIONAL clause. This query returns the *Drug_3786018* individual of the knowledge base. This is due to the drug product *Phloroglucinol* which does not contain *Dextromethorphan* nor *Zolmitriptan*. So it should not be contraindicated to *Iproniazid*.

5.3 Detection of missing data

The queries we have generated so far enable the detection of CIND violations. As stated in Section 1, we also want to consider the completeness of relation instances

involved in embedded INDs. We illustrate the search for missing values through our running example.

Example 5.4: Let us consider Table I's dataset, the SPARQL queries we have generated so far do not detect the incompleteness of the information related to the *Tuxium 30mg* drug (t4). In fact this product is composed of the *Dextromethorphan* molecule (t10) but is not contraindicated to *Iproniazid*, i.e. a tuple *(3311692, N06AF05)* is missing from the *drugContra* relation.

We have designed a solution based on the generation of SPARQL queries to detect missing values. Again this query generation exploits both the embedded IND and the pattern tableau of a CIND. The main difference lies in the direction of the embedded IND of a CIND.

Until now, the SPARQL queries we have generated exploit the traditional direction of an IND. Our detection of missing data solution generates queries based on the inverse of the embedded IND of a CIND. Investigations on exploiting inverse of rules have already been conducted. For instance, [Levy et al. 1995] presents an algorithm to answering queries using views based on the inversion of rules.

The embedded INDs of our CINDs are restricted syntactically and this supports a simple inversion plan. The inversion is performed in two steps: (1) inversion of the embedded IND and (2) inversion of the pattern tableau T_p. The first step requires to switch relations and attributes between the RHS and the LHS. The second step is also a simple inversion to the relations, attributes and tuples of T_p from the RHS to LHS and vice versa. A special attention is given on the attributes of Y_p which are containing disjunctions: each disjunct generates a new pattern tuple and all other constants and variables of the original tuple are copied. We illustrate this approach on the pharmaceutical example.

Example 5.5: The inverse of ϕ_4 yields the following embedded IND and tableau: ϕ_4^- : $atcDrug[CIP; ATCCODE] \subseteq drugContra[CIP; ATCCODE]$ and $T_4^- = \{('\_','N02CC03','\_','N06AF05'), ('\_','R05DA09','\_','N06AF05')\}$

This inverted CINDs can now be used by the SPARQL query generation presented earlier.

Example 5.6: The query generated for T_4^-'s tuple of ϕ_4^- is:

```
SELECT ?drug WHERE { ?drug rdf:type :Drug.
  ?drug :contains ?atc. ?atc :atcCode ?code.
  FILTER (?code = "N02CC03").  OPTIONAL { ?drug :contraIndicated ?atc2.
     ?atc2 :atcCode "N06AF05". }
  FILTER(!bound(?atc2)). }
```

The execution of this query on the knowledge base returns the individual identified by *Drug_3311692*. Together with the context of this query, i.e. embedded IND and database instance, this identifier provides valuable information for data repairing. Intuitively, something is wrong with the contraindication of this product.

5.4 Violation detection optimization

The number of queries that our methods generate can be very large and this impacts maintenance operations and detection performance. A first simple solution to minimize the amount of queries is to analyze the pattern tableau of a given

conditional dependencies. For example, the T_4^- tableau has 2 tuples with the same pattern : $('\_', X, '\_', 'N06AF05')$ where X is an ATC code. Hence, instead of generated 2 different SPARQL queries (one of which is displayed in Example 5.6), we can generate a single one which uses a disjunction:

```
SELECT ?drug WHERE { ?drug rdf:type :Drug.
  ?drug :contains ?atc. ?atc :atcCode ?code.
  FILTER (?code = "R05DA09" ||  ?code="N02CC03").
  OPTIONAL { ?drug :contraIndicated ?atc2.
  ?atc2 :atcCode "N06AF05". } FILTER(! bound(?atc2)).}
```

We now propose two more involved approaches to reduce the amount of queries: merging pattern tableaux whenever conditional dependencies are comparable and exploiting the structure of an ontology to optimize a set of SPARQL queries. Note that both of these approaches can be used together on a given set of conditional dependencies.

Tableau merging

Because a tableau merging has already been proposed in [Bravo et al. 2007], in this section, we focus on a novel tableau merging solution for CINDs. The merge of a pattern tableau is based on the notion of comparability of conditional data dependencies. That is the tableaux of the CINDs are extended with a set of attributes from the relations involved in the CINDs. We consider the 2 following CINDs:

ϕ_5: (R1[x1;x2] $\subseteq$ R2[y1,y2], T_5) and ϕ_6 : (R1[x3;x4] $\subseteq$ R2[y3,y4], T_6).

The CIND supporting the merged tableau correspond to ϕ_m: (R1$[X; X_p] \subseteq$ R2$[Y; Y_p]$, T_m) where $X = x1 \cup x3$, $X_p = x2 \cup x4$, $Y = y1 \cup y3$ and $Y_p = y2 \cup y4$. We then copy all the tuples of T_5 and T_6 in T_m and for each attribute A in T_m that is not filled with either a constant or a variable, we introduce: (i) a special symbol @, which denotes a *do not care* value, if A $\in X \cup Y$, (ii) a standard variable symbol (_) if A $\in X_p \cup Y_p$. Let us consider the following T_5 and T_6 tableaux:

T_5	x1	x2	y1	y2
	_	2	_	{4}

T_6	x3	x4	y3	y4
	_	12	_	{14,15}

We can now compute the merged tableau T_m:

T_m	x1	x2	x3	x4	y1	y2	y3	y4
	_	2	@	_	_	{4}	@	_
	@	_	_	12	@	_	_	{14,15}

We now need to clarify the notion of CIND satisfaction in the presence of @. This is done by distinguishing for a tuple t the subset of attributes of X that have no @ symbols. This subset is denoted by X_t^{free} (respectively Y_t^{free} for Y). Then, an instance (I_1, I_2) of (R_1, R_2) satisfies the CIND ϕ, denoted $(I_1, I_2) \models \phi$, iff for each tuple t_p in the pattern tableau T_p and for each tuple t_1 in the relation I_1, if $t_1[X_t^{free}; X_p] \asymp t_p[X_t^{free}; X_p]$ then there must exist t_2 in I_2 such that $t_1[X_t^{free}] = t_2[Y_t^{free}] \asymp t_p[Y_t^{free}]$ and moreover $t_2[Y_p] \asymp t_p[Y_p]$. Note that doing so, we treat disjunctive and non disjunctive patterns together.

This method enables to detect violations of a set of CINDs, i.e. it identifies tuples of R1 that do not have a counter part in R2. But this solution does not enable to

explain violations since many patterns are bundled in a given SPARQL query. The solution consisting of multiple SPARQL queries, i.e. one for each CIND, enables to explain violations and thus may be used to guide the end-user in cleaning the database. It is up to the creator of a data quality tool to select one of these 2 approaches, that is whether she wants to provide an explanation solution or not.

Ontology-based structure optimization

This optimization method exploits the ontology structure, i.e. mainly concept and property hierarchies, to minimize the number of generated queries. For instance, consider the Ketoprofen molecule, whose ATC code is M01AE03. In a first step, our discovery system was able to state contraindications with Vitamin K antagonists (B01AA), Heparin group (B01AB), Platelet aggregation inhibitors excluding heparin (B01AC), Enzymes (B01AD) and Direct thrombin inhibitors (B01AE). Analyzing the ontology structure, our system is able to discover that this set of ATC codes exactly covers the B01A ATC code (Antithrombotic agents). Hence, instead of generating a SPARQL query for each of these 5 codes or generating a single SPARQL with a disjunction of those 5 ATC codes, it is more efficient to create a single SPARQL query with the B01A ATC code. This means that we are delegating some of the ATC identification tasks to an ontology reasoner, e.g. the Pellet reasoners which is tailored to the OWL language.

5.5 Detection methodology

Based on the conditional dependencies discovered and the approaches proposed in this section, it is possible to develop tools that enforces the consistency of the knowledge base.

Essentially, these tools proceed in two steps: (1) detect CFD and CIND violations via the execution of a set of queries generated with the method of Sections 5.1 and 5.2. These tools can easily provide explanations about the context of the detection and identify the node of the graph responsible for the inconsistency. (2) detect missing values via the generation of SPARQL queries using the inverse of CIND approach (Section 5.3). Again, contextual information is able to guide end-users toward easy repairing of the knowledge base and thus the database.

6. EXPERIMENTAL EVALUATION

Since the data quality improvement of our approach depends on the accuracy and efficiency of the discovery of CFDs and CINDs, it is necessary to run an experimental study on these aspects. Such studies have already been conducted on CFDs so we concentrate on the CIND case. Our tests are exploiting several databases (real and synthetic). The synthetic databases are generated using real data from our drug database. Finally, we emphasize on the data quality improvements on our drug database.

6.1 Experimental settings

Real datasets: The drug database contains all drugs sold on the French market, i.e. 10000 drug products. Moreover, it contains 25 relations and 110 attributes. In this experiment, we are particularly interested on the relationship between the contraindication and the composition of drug products. The representation of this relationship with CINDs is preponderant in the development of medical application.

Moreover, this set of experiments reports on the analysis conducted with healthcare professionals on the validity of the obtained results. The graphical representation of the pattern tableaux of CINDs were particularly ease to interpret for these domain experts.

Synthetic datasets: For the purpose of this experimentation, we have designed a generator of synthetic databases consisting of an implementation of the Chase algorithm [Abiteboul et al. 1995] that satisfies a set of INDs. The synthetic databases are generated from real data coming from the medical domain. The database contains two relations which are related by an IND. The database instances we have generated vary in terms of the number of tuples and the number of attributes. Generating synthetic databases with increasing number of attributes, we experimented on increasing the arities of $|X|$, $|X_p|$, $|Y_p|$. The results we present in this section also consider the arities of X_{np} and Y_{np} with $X_{np} = R - (X \cup X_p)$ and $Y_{np} = R_(Y \cup Y_p)$. As we are generating the synthetic databases and thus program the appearance of patterns distributed on attributes of R_1 and R_2, we found particularly interesting to deliberately introduce attributes which are not containing any patterns. The generated synthetic databases all contained at least binary INDs.

The algorithms have been implemented in Java and tested on an Intel Centrino Duo Processor (2.6GHz) with 4GB of memory running the Linux operating system. The relational database management system we have used is PostgreSQL 8.3. All performance measurements reported represent averages of five trials.

6.2 Experimental results

We have designed synthetic databases according to the following parameters: number of tuples ranging from 100K to 1000K, number of attributes between 12 and 28 and a minimal support value ranging from 5 to 100. We analyze results in terms of scalability w.r.t. the Tuple and the Attribute parameters. In fact, they represent the amount of time spent by both algorithms to compute a set of CINDs. Other experiments concentrate on the Support parameter by studying the number of CINDs discovered and the time required for our discovery approach when the support threshold increases.

Scalability w.r.t. Tuple: We study the scalability of our algorithm with respect to the number of tuples in both relations. The relations of all the synthetic databases in this experiment contain 6 attributes for the relation on the LHS of the IND and 7 attributes for the RHS. The minimal support value is set to 5 and the number of tuples are ranging from 100K to 1000K. The results presented on Figure 1(a) shows that the algorithm scale linearly for the metrics considered.

Scalability w.r.t. Attribute: We now study the effect of increasing the number of attributes in both relations involved in the embedded IND. The database we have selected corresponds to a synthetic one with two relations and a binary IND between them. The support value is set to 5 and each relation contains 200K tuples. We vary the number of total number attributes from 12 to 28 and decompose it such that both relations have the same number of attributes. Figure 1(b) shows that the algorithm scale linearly for the metrics considered

Number of CINDs discovered w.r.t. Support: Figures 1(c) and 1(d) displays the results of varying the support from 5 to 100 on respectively the synthetic database of 200K and the drug database. As expected, the lower the support, the

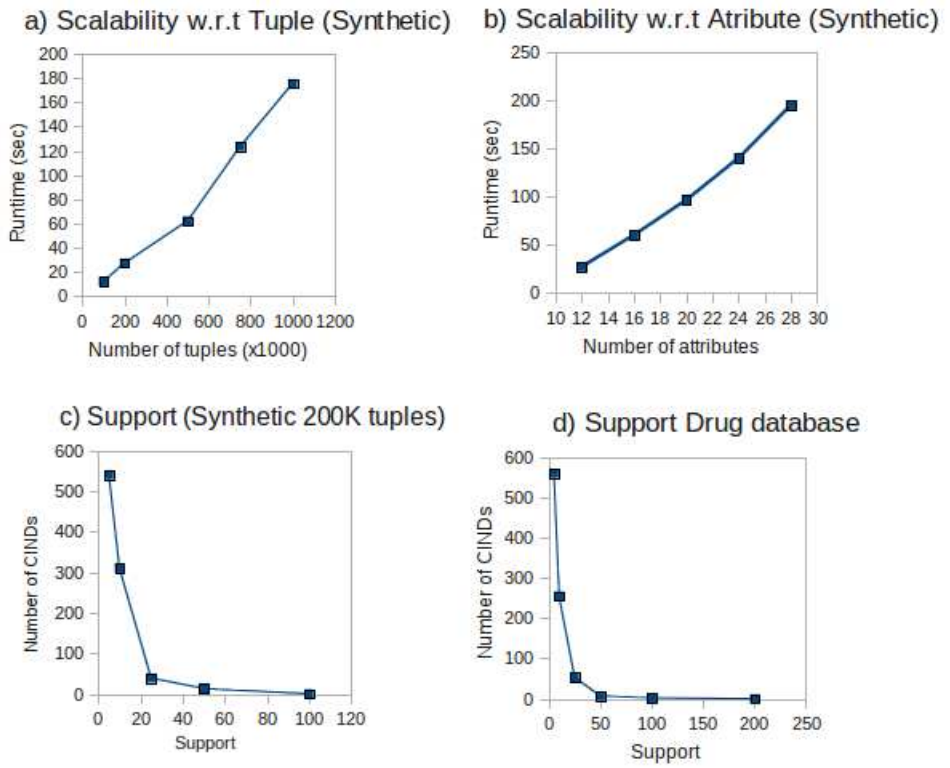

Fig. 1. Scalability and accuracy for CINDs discovery

more CINDs are discovered. This is explained by the domain of the database and the semantics of the attribute supporting the embedded INDs.

Analysis of the discovered CINDs: The analysis of the relevancy of the discovered CINDs of the *drug* database has been conducted with the team of healthcare professionals working on our self-medication application. We presented them the pattern tableaux found on the contraindication experimentation. These domain experts, with no peculiar expertise in database technology, appreciated the visualization possibilities offered by the tableaux and in particular on the representation of disjunctions. This feature is particularly important on this use case since molecules are generally contraindicated with many other molecules and many diseases.

The discovered CINDs are evaluated in terms of soundness and completeness. Concerning soundness, all contraindication patterns discovered are sound, that is no false contraindication has been proposed by the system. Again, the generation of such a repository of pharmaceutical information is quite important since it is absent from the standards generally provided. In terms of completeness, some real-life contraindications are missing from the set discovered. After an investigation of the drug database, it showed that the responsibility originates from the sample database and not the discovery method. This raises the issue of the quality (soundness and

completeness) of the database supporting the discovery. The measures proposed in [Chiang and Miller 2008] provides a preliminary approach toward addressing this issue.

This data quality tool is now used in a real world setting which serves a medical informatics application. In this application, an important number of drug information have been processed toward conditional dependencies discover, e.g. side-effects, drug and disease contraindications and cautions. The CFDs and CINDs discovered are stored in a repository and their discovery are processed on a monthly basis. The drug database stores almost all products of the French market (around 10,000 entries) and concern more than 2,000 molecules. Intuitively, the data quality aspect of the application works as follows. Drug product updates are stored as SQL queries in a text document. These queries enable to remove, insert or modify a set of tuples of the database. Regularly, this text document is sent to the database. During this processing, our data quality solution executes the stored SPARQL queries to detect inconsistencies and missing values. Each CFD of CIND violation is cured with the help of a health professional who corrects the dirty data directly either at the knowledge or database levels. (due to a synchronization solution provided by the DBOM system [Curé and Squelbut 2005]). After several months of using this data quality approach, we consider that we have improved the overall quality of our database by 15%. This has been calculated by a ratio of the total number of data items modified on the total number of data items stored in our database. Moreover, the more accurate becomes our database, the more relevant the generated set of CFDs and CINDs.

7. IMPLICATIONS TO RESEARCH AND PRACTICE

We consider that applying the different contributions presented in this paper can have a significant impact on the work of researchers and practitioners involved in data quality enhancement. The availability of CFD and CIND discovery algorithms opens up a new field for research and practice in data cleansing.

For practitioners, these algorithms can be easily implemented on any programming language and executed over their database instances. Then the discovered conditional dependencies can be integrated as SQL or SPARQL queries in their applications. Developers with applications involving ontologies are encouraged to select a SPARQL approach since functionalities like violation explanations can easily be programmed using OWL reasoners, e.g. Pellet, and OWL APIs, e.g. OWLAPI and Jena.

Concerning researchers, we consider that many studies need to be conducted on the cooperation between practical CFDs and CINDs. The upcoming SPARQL 1.1 recommendation will also open research tracks due to its support for an update language for RDF graphs, enrichment with aggregates, subqueries, projected expressions, negation and use with entailment regimes.

First of all, the techniques presented in this paper are domain independent. That is, researchers and practitioners do not need to be involved in medical projects or drug databases to apply our contributions. We consider that due to the availability of ontologies in many scientific domains, the data quality of scientific databases can be enhanced using our conditional dependency approach. It may also be applied in

sociological and cultural domains where structured terminologies are also emerging (e.g. FOAF and CRM CIDOC projects).

8. CONCLUSIONS

The data quality of medical databases is predominant to many medical informatics application. The approach adopted in this paper consists in exploiting a novel form of database dependencies where constant values are supported in tableau patterns. Recently, several papers investigated the discovery of conditional dependencies but these research concentrated on CFD and no proposition was available for CIND. This paper proposes a first solution to this problem which is quite important since it is considered that the full potential of data quality tools can be reached within the framework of conditional dependencies when both CFD and CIND are supported. Apart from studying this aspect of CIND, we also extended their expressiveness by introducing disjunctions on the right hand-side of the embedded IND. Following the path drawn by CFD research, this paper proposes solutions to represent these dependencies in a declarative way and to minimize their number by merging pattern tableaux.

Nevertheless, our approach differs from previous work by integrating an ontology approach to represent domain knowledge and constraints. We believe that this is particularly relevant with the emergence of open data in various domains, e.g. geography and medicine. Moreover, many functionalities, such as reasoning and entailment explanations, are natively supported in some knowledge base contexts but are very hard or impossible to implement in a relational database context. Because of this ontology integration, queries representing conditional dependencies are serialized in SPARQL, a query language adapted to RDF graphs. The main benefit of using this approach consists in minimizing a set of queries by exploiting the (subsumption) reasoning facilities of a description logics reasoner.

This data cleaning solution has enhanced the information accuracy of our drug database in the context of a self-medication application. So far, healthcare professionals are heavily involved in cleaning the inconsistent database tuples. We believe that this is the safest way to handle the many exceptions holding in the medical domain. But we consider that some of these data repairs can be automatized by studying the context of the inconsistency.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., Eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Batini, C. and Scannapieco, M. 2006. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bohannon, P., Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. 2007. Conditional functional dependencies for data cleaning. In *ICDE*. IEEE, 746–755.

Bravo, L., Fan, W., Geerts, F., and Ma, S. 2008. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*. IEEE, 516–525.

Bravo, L., Fan, W., and Ma, S. 2007. Extending dependencies with conditions. In *VLDB*. 243–254.

Chiang, F. and Miller, R. J. 2008. Discovering data quality rules. *PVLDB 1,* 1, 1166–1177.

Curé, O. 2004. Ximsa : extended interactive multimedia system for auto-medication. In *CBMS*. 570–575.

Curé, O. and Bensaid, J.-D. 2008. Integration of relational databases into owl knowledge bases: demonstration of the dbom system. In *ICDE Workshops*. 230–233.

Curé, O. and Squelbut, R. 2005. A database trigger strategy to maintain knowledge bases developed via data migration. In *EPIA*. 206–217.

Fan, W. 2008. Dependencies revisited for improving data quality. In *PODS*, M. Lenzerini and D. Lembo, Eds. ACM, 159–170.

Fan, W., Geerts, F., Laksmanan, L. V., and Xiong, M. 2009. Discovering conditional functional dependencies. In *ICDE*. To appear.

Giroud, J.-P. and Hagege, C. 2001. *Le guide de tous les médicaments*. Editions du Rocher Paris, France.

Goethals, B., Page, W. L., and Mannila, H. 2008. Mining association rules of simple conjunctive queries. In *SDM*. SIAM, 96–107.

Golab, L., Karloff, H. J., Korn, F., Srivastava, D., and Yu, B. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB 1,* 1, 376–390.

Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. 1995. Answering queries using views. In *PODS*. ACM Press, 95–104.

Marchi, F. D. and Petit, J.-M. 2003. Zigzag: a new algorithm for mining large inclusion dependencies in database. In *ICDM*. IEEE Computer Society, 27–34.

Mitchell, J. C. 1983. The implication problem for functional and inclusion dependencies. *Information and Control 56,* 3, 154–173.